

Efficient transmission of measurement data from FPGA to embedded system via Ethernet link

Wojciech M. Zabolotny*

Institute of Electronic Systems, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland

Abstract

This paper presents a system consisting of the FPGA IP core, the simple network protocol and the Linux device driver, capable of efficient and reliable data transmission from a low resources FPGA chip to the Linux-based embedded computer system, via a private Ethernet network (consisting of a single segment or a few segments connected via an Ethernet switch). The embedded system may optionally process the acquired data, and distribute them further, using standard network protocols.

Proposed design targets cost-efficient multichannel data acquisition systems, in which multiple FPGA based front end boards (FEB) should transmit the stream of acquired data to the computer network, responsible for their final processing and archiving.

The presented solution allows to minimize the cost of data concentration due to use of inexpensive Ethernet network infrastructure.

The work is mainly focused on minimization of resources consumption in the FPGA, and minimization of acknowledge latency in the Linux based system - which allows to achieve high throughput in spite of use of inexpensive FPGA chips with small internal memory.

Keywords: FPGA, Ethernet, Ethernet Protocol, Embedded Systems, Data Acquisition, Data Concentrator

1. Introduction

Contemporary measurement systems often are spatially distributed and use multiple input channels to acquire data. The designers of the data acquisition (DAQ) part of such system have to solve the problem how to transmit data from multiple front end boards (FEB), receiving signals from the sensors and converting them into the digital form, to the data processing center (often a computer grid or storage array), which will finally process and archive those data (see Figure 1). This process involves concentration of data, often associated with preprocessing or aggregation of concentrated data. In systems with high numbers of input channels, the cost of the data concentration subsystem may significantly affect the cost of the whole DAQ, and therefore development of cost-efficient solutions is desirable.

The digital part of FEB boards is often based on the Field Programmable Gate Arrays (FPGA) chips. Such solution offers many advantages:

- high flexibility, and possibility to adjust or correct data acquisition algorithms without hardware modifications
- good performance of simultaneous processing of parallel data streams

*Tel. +48 22 234 7717; fax.: +48 22 825 2300

Email address: wzab@ise.pw.edu.pl (Wojciech M. Zabolotny)

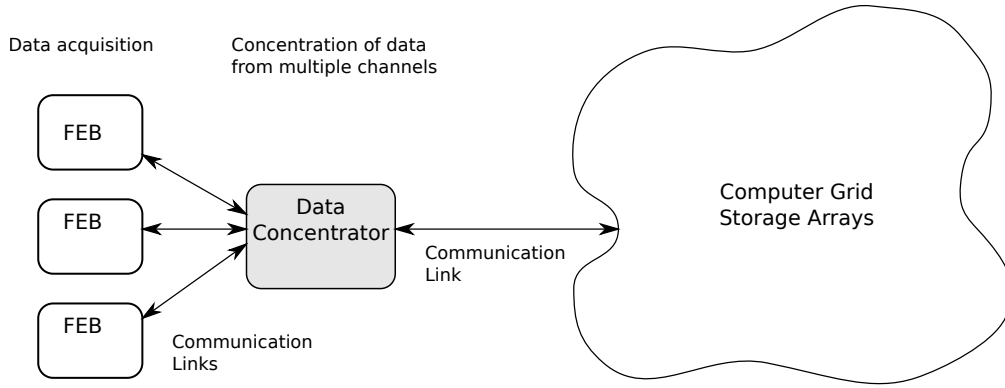


Figure 1: Data acquisition system with multiple FEBs connected via data concentrator to the data processing grid.

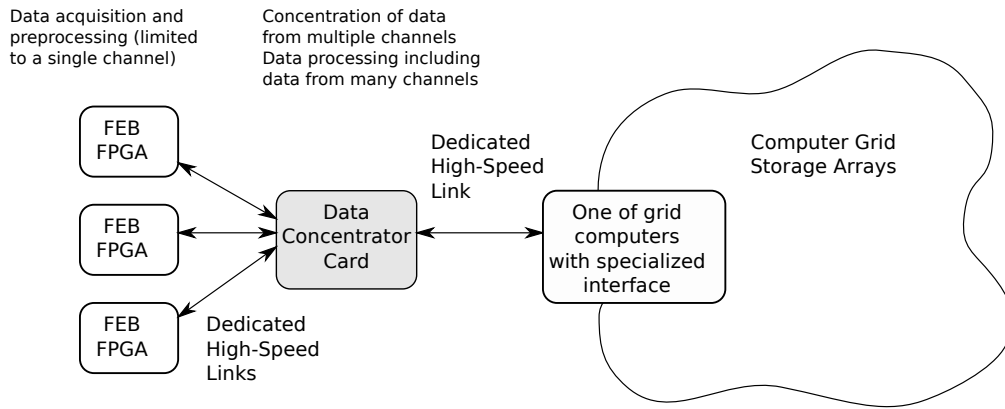


Figure 2: Data acquisition system with multiple FEBs connected via dedicated high-speed links to the specialized DCC board, and further, via another high-speed link to one of computers in the data processing grid.

- easy interfacing to different digital interfaces, used to connect analog to digital converters (ADC) or sensors with digital outputs.
- deterministic timing and data sampling with minimized jitter

Because the FEBs are usually the most numerous components of the DAQ, it is desirable to minimize the cost of FPGA chips used in a FEB.

Sometimes the process of data concentration may involve computation intensive data processing using highly parallelized algorithms, and in this case the FPGA based Data Concentration Card (DCC) may be needed (see Figure 2). In this situation the high speed serial links (currently available in more advanced FPGA chips[1, 2]) may be a good solution to provide transmission of data from the FEBs to the dedicated DCC. An example of such approach may be the DCC board used in the Resistive Plate Chamber Pattern Comparator Muon Trigger in the CMS Experiment[3, 4]. The problem of such approach however is relatively high cost of development and manufacturing of such specialized DCC board. The cost may be even further increased by high cost of specialized components needed to provide connection between the FEBs and DCC.

In some cases (e.g. when only data concentration is needed, or when the preprocessing of concentrated data involves mainly sequential algorithms), there is no advantage to use the DCC card, and it may be replaced with an embedded computer system with performance suitable to handle the expected data stream. Because such embedded systems may be equipped with multiple network interfaces, and because it is possi-

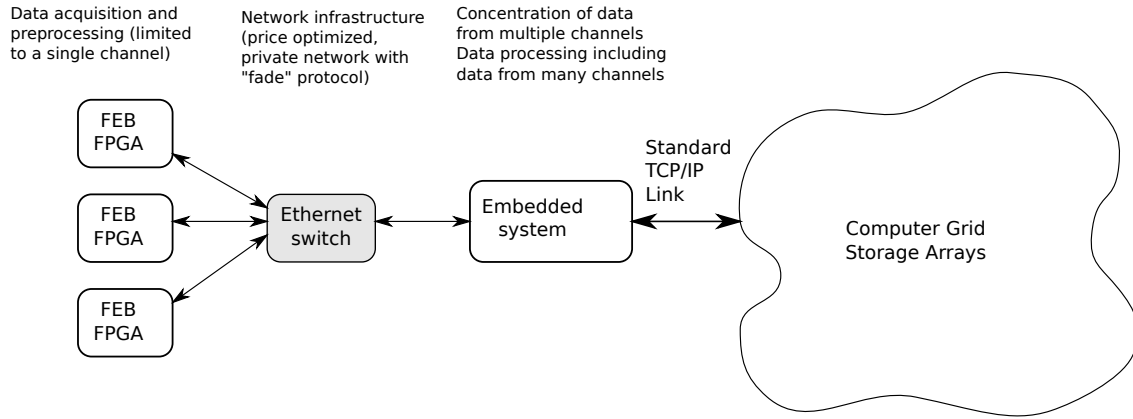


Figure 3: Data acquisition system with multiple FEB connected via Ethernet switch to a single embedded system, transmitting data to the computer grid for further processing or storage.

ble to connect the FPGA in FEB to the network physical layer interface chip (PHY), the data concentration may be performed using a standard network connection. If the throughput of the network link is sufficient, we can use the network switch to connect multiple FEBs to a single network interface.

Such approach, based on widely available, ready to use hardware solutions (network cables, network switches, network capable embedded systems) may significantly decrease the total cost of the system, and shorten the development time.

The most popular standard used in local area networks (LAN) is the Ethernet, and therefore we have focused on application of the Ethernet network to build a link between FEB boards and the computer system. Of course this computer system may further distribute the concentrated data to the whole data processing network, using the standard network interface as shown in the Figure 3.

2. Use of Ethernet interface with FPGA chips

The Ethernet interface assures high throughput and reasonable latency, however it does not assure reliable transfer of data. In typical applications Ethernet is used as a network interface to provide low-level communication, and reliability of the communication is assured by the higher layers of the network protocol.

The most popular protocol for reliable transmission of data through unreliable network connection is TCP/IP, but for our application it is unacceptable. The implementations of TCP/IP stack in the FPGA require a lot of resources [5, 6], as it is necessary to provide CPU functionalities, and additionally a lot of memory to buffer the data.

The benefits of using TCP/IP, such as routability and possibility to transmit the data via heterogeneous network (implemented in the IP layer, the 3rd layer in the OSI model) are useless for this specific application, where we need to transmit data in a single network segment or in a few segments connected via Ethernet switch.

To find the optimal solution, we must quickly analyze consequences of requirement of reliable transmission.

2.1. Reliable transmission

The Ethernet link is not reliable due to following reasons:

- Data corruption in the physical link (the bit error rate (BER) may be up to 10^{-8} in 10Base-T [7] and up to 10^{-10} in 100Base-T [8] and 1000Base-T Ethernet [9])

- Packet corruption due to collision (only in certain kinds of physical connections)
- Dropping of packets by the receiving system due to overload

Infrequent random loss of a single packet (e.g. due to noise or collision) can be mitigated by using Forward Error Correction [10] techniques, implemented on the protocol level. The simplest (and therefore implementable in a FPGA) method could be grouping of packets in N -packet groups and sending an additional *parity packet*, filled with data calculated as exclusive-or of data contained in normal packets. If any single packet in the group is corrupted (which can be detected using the packet's checksum), it may be reconstructed from other packets and the *parity packet*¹. Such a solution may be worth of checking for a full-duplex connection between single FEB and the network adapter in the computer system, as such configuration minimizes packet collisions. Unfortunately it may be not effective in a typical situation where we want to concentrate data (with a few FEBs connected via network switch to the network adapter in the computer system), as in this setup there is a higher risk of dropping of more packets from a group.

In such situation, the only way to assure reliability is to use the acknowledge/retransmission mechanism similar to the one used in the TCP/IP protocol. The problem with such solution, however, is that it requires significant amount of memory to buffer the transmitted, and not confirmed yet data.

To fully utilize the throughput of the network link, when waiting for the acknowledgment of the particular packet, we should transmit next ones. However all unacknowledged packets should be stored in the memory, because they may need to be retransmitted if no acknowledgment is received.

If we denote the transmission speed as R_{transm} , and the maximum latency of acknowledgment as $t_{ack\ max}$, then the required capacity of the memory buffer M_{buf} is given by the simplified formula (the formula does not take into account the length of the packet):

$$M_{buf} = R_{transm} t_{ack\ max} \quad (1)$$

Currently the typical amount of internal memory in inexpensive FPGA chips is below 100 KiB, and if we consider the Ethernet protocol overhead, we can state that for transmission speed of 100 Mb/s we need maximum acknowledge latency below c.a. 7 ms, and for transmission speed of 1 Gb/s below c.a. 700 μ s.

The acknowledge latency measured for TCP/IP with direct connection (via switch only) is typically below that value. The measured mean ACK latencies were:

- 170 μ s for communication with Intel Core 2 T5500/1.66 GHz based system
- 240 μ s for communication with Pentium 4/2.8 GHz based system
- 520 μ s for Ralink RT3350/320 MHz based system.

However the latency increases above 1 ms, when routing is involved.

These facts show, that if we want to assure reliable transmission from FPGA with low amount of internal memory, we can't use routing. The acknowledge must be generated by the receiving host which is connected either directly, or at most via Ethernet switch. However in this situation we also don't need an IP layer, as for communication between devices connected via Ethernet switch, the MAC addressing is sufficient.

Summarizing the above analysis:

- We can use simple, unroutable protocol using the MAC addressing.
- We should concentrate on minimizing of the acknowledge latency.

¹Similar solution was used to protect configuration data in the FLASH memories against radiation induced corruption in the Linkbox Control System for RPC subdetector in the LHC experiment [11].

If the data received from the FPGA based FEBs should be transmitted further (see Figure 3), we can use the standard network solutions for that next stage of transmission (of course the required link throughput for this connection may be higher, than for links from FEBs). As the typical Linux based embedded computer system is equipped with RAM memory with capacity of at least 64 MiB and often above 1 GiB, it is not a problem to buffer much higher amount of data than in the FPGA. Therefore, according to the formula (1), further transmission may be performed with routable protocols with higher acknowledge latency, like TCP/IP.

2.2. Avoiding of network congestion

As it was stated above, one reason why Ethernet does not assure reliable transmission, is the danger of packet loss due to collisions or due to dropping of packets by overloaded switch or receiving system. Systems, which use packet acknowledge mechanisms are prone to the network congestion problem, when quick resending of not acknowledged packets leads to increase of network load and receiver load, which in turn further increases the risk of packet loss.

To avoid the network congestion, the system should be equipped with means to monitor the ratio of the lost and retransmitted packets and to adjust the rate of sending of packets, so that the amount of dropped packets is reasonable. As the implementation described in this paper is only a “proof of the concept” solution, we have proposed very simple mechanism (described in subsection 3.2.1), where the delay between packets is adjusted depending on the ratio of lost packets. Further research is needed to find the optimal method to avoid switch or receiver overload. The problem maybe especially important in triggered data acquisition systems, where all FEBs may start to transmit data simultaneously, after the trigger is received and the network load is fluctuating.

2.3. Additional assumptions simplifying the design

The system is supposed to work over the private, physically protected network. Therefore we don't need all the features, which are introduced in protocols like TCP/IP to assure secure communication (e.g. we can use simple sequential numbering of packets). Additionally at this, initial, state of development there was no need to officially allocate the Ethernet protocol number, and an arbitrarily chosen number *0xfade* was used.

3. Implementation of the system

Next sections describe the implementation of all parts of the proposed system, including the proposed communication protocol, the FPGA IP core, and the Linux kernel driver for the embedded system.

3.1. Proposed communication protocol

Design of the communication protocol heavily depends on the details of FPGA implementation (see section 3.2) and of software implementation (see section 3.3). Therefore describing the protocol we will often mention details which are fully explained in the next sections.

The communication protocol is kept as simple as possible. Hence we use only four kinds of packets (see Table 1.)

To start and stop transmission of data from the FEB, the receiving computer system sends appropriately the START or STOP packet.

After the transmission is started, the FEB starts to send the stream of data. Data bytes are encapsulated in DATA packets containing 1024 bytes of data and an additional information (described further).

The length of packet was chosen so, that it is shorter than the Ethernet Maximum Transmission Unit (MTU), equal to 1518 bytes[7], and is equal to power of two, which allows efficient storage of packet contents in the memory buffer.

Table 1: Structure of the packets used by the transmission protocol. (SRC and TGT - MAC addresses of the transmitter and of the receiver)

packet	direction	structure							
START packet	to FEB	<table><tr><td>SRC</td><td>TGT</td><td>0xfade</td><td>0x0001</td><td colspan="3">padding to 64 bytes</td></tr></table>	SRC	TGT	0xfade	0x0001	padding to 64 bytes		
SRC	TGT	0xfade	0x0001	padding to 64 bytes					
STOP packet	to FEB	<table><tr><td>SRC</td><td>TGT</td><td>0xfade</td><td>0x0005</td><td colspan="3">padding to 64 bytes</td></tr></table>	SRC	TGT	0xfade	0x0005	padding to 64 bytes		
SRC	TGT	0xfade	0x0005	padding to 64 bytes					
DATA packet	from FEB	<table><tr><td>SRC</td><td>TGT</td><td>0xfade</td><td>0xa5a5</td><td>set & packet number</td><td>delay</td><td>1024 bytes of data</td></tr></table>	SRC	TGT	0xfade	0xa5a5	set & packet number	delay	1024 bytes of data
SRC	TGT	0xfade	0xa5a5	set & packet number	delay	1024 bytes of data			
ACK packet	to FEB	<table><tr><td>SRC</td><td>TGT</td><td>0xfade</td><td>0x0003</td><td>set & packet number</td><td colspan="2">padding to 64 bytes</td></tr></table>	SRC	TGT	0xfade	0x0003	set & packet number	padding to 64 bytes	
SRC	TGT	0xfade	0x0003	set & packet number	padding to 64 bytes				

The data stream is logically divided into *data sets*. One *data set* is a group of consecutive packets, carrying the data, which fit into the memory used to buffer the transmitted and not confirmed yet packets. Due to the technical characteristics of FPGA platforms available for tests (see section 4), the length of the *data set* was chosen as $N_{pkts} = 32$. The number of packets in a set is also a power of two, which simplifies addressing of the memory.

The transmitted packets are labeled with the sequential number, created by the concatenation of the 16-bit set number and 5-bit packet number²

After reception of the DATA packet, the receiving system confirms the successful reception, sending the ACK packet, labeled with the same set number and packet number as the acknowledged packet.

The proposed protocol may be extended with additional command packets, sent from the embedded system to the FPGA, providing more advanced control of the FEB's operation. Such extension, however, implies necessity to introduce the acknowledge/retransmit mechanism for the command packets as well. (Currently implemented commands START and STOP do not require acknowledgement, as their correct reception is confirmed by sending or not sending of data by the FEB).

3.2. FPGA implementation

Structure of the FPGA IP core is shown in the figure 4

The input of the IP core behaves like typical FIFO input. The *dta_ready* signal informs if the core is ready to accept the new data. The *dta* signal is a 32-bit wide data bus. The *dta_we* signal is the data write strobe.

Main part of the IP core is the subsystem which manages transmission and retransmission of packets (the *Descriptor Manager* - further denoted as DM), and stores the packets (the *Packet Buffers Memory* - further denoted as PBM).

The PBM, used to buffer the transmitted data, is divided into $N_{pkts} = 32$ packet buffers, each 1024 bytes long. The PBM works as a circular buffer. The i^{th} packet of any set is always stored in the i^{th} packet buffer. The PBM always stores packets belonging to one *data set* or to two neighboring *data sets* (see Figure 5).

² In fact current FPGA sources reserve place for 6-bit packet number, as some of tested FPGA chips allow to use longer sets. Additionally for debugging purposes it is possible to include the 10-bit retry number (currently not used) in each packet.

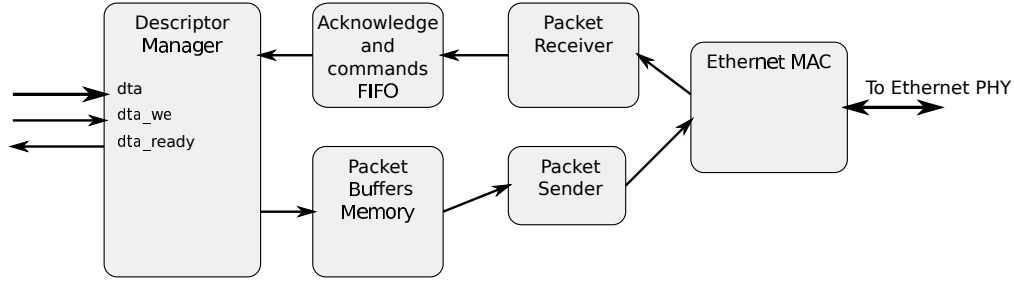


Figure 4: Structure of the FPGA IP core implementing the hardware part of the system.

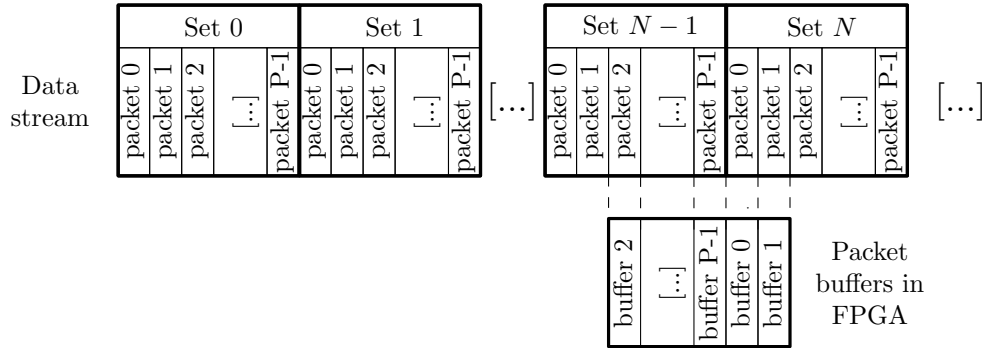


Figure 5: Relation between packets and packet buffers. Each packet set contains $P = N_{pkts}$ packets numbered from 0 to $P - 1$.

Information about the state of each packet buffer is stored in the corresponding *packet descriptor* record stored in the DM.

Each descriptor stores:

- three bit flags, describing the state of the packet:
 - *valid* (V) - set, when the packet is filled with the data,
 - *sent* (S) - set, when the packet has been transmitted at least once,
 - *confirmed* (C) - set, when the reception of the packet has been confirmed by the receiver.

Initially all those flags are set to 0.

- The *set number* - informing to which *data set* belong the data stored in the particular packet buffer

The table of *packet descriptors* is handled as a circular buffer by the DM. The *head* pointer points to the packet buffer which is currently being filled with the data. The *tail* pointer points to the last filled packet buffer, which has not been confirmed yet by the receiver, or is equal to the *head* pointer, if there are no packet buffers ready for transmission. The third *retr* pointer points to the packet buffer, which should be transmitted or retransmitted when the transceiver is ready, and when the required delay between packets (see section 3.2.1) has elapsed.

The main state machine in the DM handles three different tasks (in order of decreasing priority):

- reception of acknowledge packets, and handling of *tail* pointer
- detection of completely filled packets and handling of *head* pointer

- transmission and retransmission of packets and handling of *retr* pointer

Whenever the acknowledge packet is received, the state machine checks if the *set number* in the acknowledge packet agrees with the *set number* in the corresponding packet descriptor. If the *set numbers* do not agree, such acknowledge packet is discarded. If the *set numbers* agree, the C flag is set to 1. Additionally, if the acknowledged packet buffer is the one pointed by the *tail* pointer, we start moving the *tail* pointer, until it points to the first sent but not confirmed packet ($S=1$ and $C=0$) or is equal to the *head* pointer (meaning that all data are transferred).

If the new data are delivered to the input, they are written to the packet buffer pointed by the *head* pointer. When this packet buffer is completely filled, it is marked as ready for transmission (by setting $V=1$), and the signal *dta_ready* is cleared, signaling, that the DM is not ready for new data.

The state machine detects this state, and tries to move the *head* pointer to the next position. If the next position is still pointed by the *tail* pointer, it means, that there is no place for new data, and we must wait until the data are transmitted and confirmed. If the next position is free, the *head* pointer is moved and the signal *dta_ready* is set, allowing to feed the DM with the next data.

Whenever the *head* pointer is moved, the flags V, S and C in the descriptor pointed by the new *head* value are cleared, and the *set number* in this descriptor is set to the number of currently transmitted *data set*.

Finally the state machine checks, if the *retr* pointer points to the packet buffer which is valid, but not confirmed ($V=1$ and $C=0$). If yes, it orders transmission or retransmission of the corresponding packet, sets the S flag in its packet descriptor, and moves the *retr* pointer to the next packet descriptor ready for transmission or retransmission ($V=1$ and $C=0$). If *retr* pointer reaches the *head* position it is wrapped to the *tail* position. If the buffer is empty ($head = tail$) no transmission is attempted³.

Blocks *Packet sender* and *Packet receiver* are designed to autonomously service transmission and reception of the data. They are introduced to parallelize reception of the data from the system input (associated with packet buffers management), and Ethernet transmission. Additionally those blocks (together with the *Acknowledge and Commands FIFO*) allow to separate clock domains between the Ethernet related part of design and the rest of the system.

To control the Ethernet PHY our system uses the open source Ethernet MAC [12], however it can be also easily adapted to work with another MAC or to directly communicate with the Ethernet PHY.

3.2.1. Implementation of network congestion avoidance in FPGA

To avoid network congestion, caused by too high frequency of packet transmission, resulting in dropping of packets by the switch or by the receiving system, we have introduced adjustable delay between transmitted packets.

Because selection of the proper delay in advance is difficult or even impossible, we have introduced simple mechanism to adapt this delay.

Transmission starts with the delay set to the high value, equal to $200\mu s$, which should minimize number of lost packets, at cost of suboptimal utilization of the link throughput. During transmission the FPGA measures the ratio between the number of all transmitted packets and the number of retransmitted packets and adjusts the delay accordingly.

Whenever a packet is sent or resent, we increase the counter of sent packets ($C_{pkt\ sent}$). If the transmitted packet is resent ($S=1$ in descriptor flags) additionally we increase the counter of resent packets ($C_{pkt\ rsnt}$).

After the pre-defined number of packets is sent ($C_{pkt\ sent} = N_{pkt\ update}$), we check the ratio of the resent packets $R_{rsnt} = C_{pkt\ rsnt} / C_{pkt\ sent}$, and compare it with two pre-defined thresholds: T_{high} and T_{low} .

³This simplified description does not cover special situation, when just confirmed packet is the one pointed by the *retr* pointer. In this situation the *retr* pointer must be also modified.

If the ratio of resent packets is higher then T_{high} , the delay between packets is multiplied by the factor $\alpha_{incr} > 1$. If the ratio of resent packets lower then T_{low} , the delay between packets is multiplied by the factor $\alpha_{decr} < 1$. Afterwards both counters ($C_{pkt\ rsnt}$ and $C_{pkt\ sent}$) are cleared.

Tests were performed for two different sets of values of the parameters:

- $N_{pkt\ update} = 3000$, $T_{high} = 1/16$, $T_{low} = 1/64$, $\alpha_{incr} = 1.25$, $\alpha_{decr} = 0.9375$.
- $N_{pkt\ update} = 10000$, $T_{high} = 1/8$, $T_{low} = 1/32$, $\alpha_{incr} = 1.25$, $\alpha_{decr} = 0.75$

In tests (see section 4) both sets of settings provided reliable operation.

3.3. Implementation of the Linux driver

In the embedded system the communication with FEBs is handled by a device driver, working in the Linux kernel space, which may be controlled by the user space application. The user space application may receive acquired data in an efficient way, using the memory mapped circular buffer. Such approach simplifies implementation of algorithms of data preprocessing and further distribution, as the user space code is easier to develop and debug than the kernel code. Use of a memory mapped buffer, instead of traditional socket interface, to access received data from the user space, allows to decrease overhead needed to handle data.

The driver may service multiple FPGA based FEBs, and multiple network interface cards (with possibility to have a few FEBs connected via a switch to one network card). The maximum number of FEBs is defined by the module parameter *max_slaves*. For each FEB a separate character device (*/dev/l3_fpga0*, */dev/l3_fpga1* and so on) is created with separate circular buffer. The data in each buffer may be accessed directly using *mmap* technique, but the pointers position in the particular buffer must be accessed using the *ioctl* function, to assure proper synchronization. Such approach assures both fast and secure access to the received data.

The main component of the device driver is the protocol handler, installed via *dev_add_pack* function [13, chap. 13], which is called whenever the Ethernet frame with *Oxfade* type is received. The protocol handler first checks if the received packet has been transmitted by the registered and started FEB (and if no, it sends the STOP command to the source of the packet). Then it checks if the received packet has reasonable *set number*. If the *set number* corresponds to the already confirmed packet, the acknowledgment is sent immediately to handle situation when the previous acknowledge packet got lost. If the *set number* corresponds to the packet, which has not been received yet, the handler tries to copy received data to the circular buffer, and if it succeeds, marks the packet as confirmed and sends the acknowledgment packet.

Due to possible loss of packets, it is not warranted, that the data arrive in sequence. Therefore the *head* pointer is moved only to the end of the continuous area filled with the received data.

To speed up reception of the data, some optimizations are undertaken. The length of the circular buffer is a multiple of the length of single *data set* ($n \cdot N_{pkts} \cdot 1024$ bytes), and therefore the data associated with particular *data set* always occupy a continuous area in the circular buffer. As we always expect packets only from two consecutive *data sets* (see Figure 5), it is enough to maintain two pointers, pointing to the beginning of data associated with those *data sets* in the circular buffer.

Described optimizations allow to minimize the time needed to copy the data and to acknowledge the packet, which in turn allows to achieve faster transmission, according to formula 1.

3.3.1. Communication with the user space application

The user space application may connect to one or more FEB devices, opening one or more created character devices (*/dev/l3_fpga0*, */dev/l3_fpga1* and so on), and mapping its circular buffer into the application's address space.

The application may connect to the particular FEB, and start transmission, using the special *ioctl* call: *L3_VI_IOC_STARTMAC*, passing address of the structure describing the desired FEB device (containing its MAC address, and the name of the network interface - e.g. "eth0").

After the FEB device is connected, the system will start to receive the data, writing them to the circular buffer. The application may read the *head* and *tail* pointers for the particular FEB device with another ioctl call `L3_V1_IOC_READPTRS`, which additionally returns the number of available bytes of data. Then the application may process the data located between the *tail* and *head* position. After data are processed, the application may confirm their reception, calling the `L3_V1_IOC_WRITEPTRS` ioctl with number of processed bytes.

Data located between the *tail* and *head* positions are warranted to be unchanged, so they may be safely read and processed by the application, while the *ioctl* function takes care of proper synchronization of access to the pointers between the application and the protocol handler.

To optimize use of the CPU, the application may sleep, waiting for the data. To allow servicing of multiple FEB devices, the sleep functionality has been implemented in the *poll* function. The number of available received bytes, needed to wake up the application may be defined with ioctl (`L3_V1_IOC_SETWAKEUP`) call.

Additional ioctl commands allow to stop the transmission from particular FEB device (`L3_V1_IOC_STOPMAC`), and to read the total length of the particular circular buffer (`L3_V1_IOC_GETBUFLLEN`).

3.3.2. Code portability

The Linux device driver was prepared for Linux kernels 3.3.x , but it compiles also with newer kernels. Particularly it has been successfully compiled and used with Linux kernels 3.4.4 (in Knoppix 7.0.3 [14]), 3.5.2 and 3.5.3 (with Debian/testing). The code has been implemented in a multiprocessor-safe way.

4. Tests, results and discussion

The system was tested using the Dell Vostro 3750 (Intel Core i7-2630QM CPU, 2.0 GHz clock) computer running the Debian/testing Linux OS (simulating the embedded system). Use of computer with 4-core CPU (with hyperthreading capable cores) allowed to confirm, that the code works reliably in multiprocessor environment. The FPGA based FEBs were simulated with three evaluation boards:

- SP601 evaluation board[15] equipped with 10M/100M/1G Ethernet PHY
- Atlys board[16] equipped with 10M/100M/1G Ethernet PHY
- Spartan-3E Starter Kit[17] (further denoted as S3ESK) equipped with 10M/100M Ethernet PHY

4.1. Results of FPGA compilation

The IP core was successfully compiled for all FPGA platforms used for tests. The resources consumption for different platforms is shown in the Table 2. As can be seen, in all tested FPGAs, synthesis of our IP core leaves significant amount of resources for another, user defined functionalities.

Probably resources consumption may be further decreased by replacement of the OpenCores MAC core [12] with a simplified core, communicating directly with the Ethernet PHY. Unfortunately such implementation is yet not mature enough to be published.

4.2. Results of transmission tests

The tests were performed using a simple application, which received data sent by the emulated FEBs and immediately confirmed their reception, freeing the buffer.

The tests covered:

- measurement of the throughput

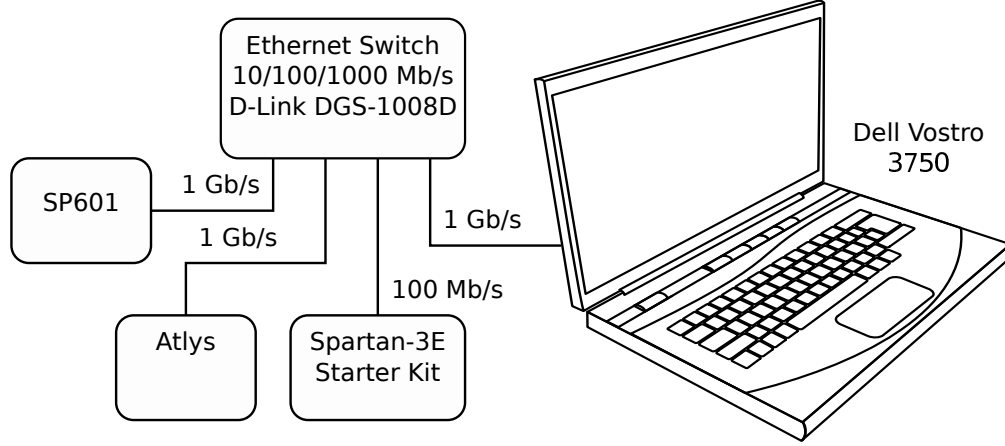


Figure 6: Test setup.

Table 2: Results of compilation of the IP core for different platforms

Board	FPGA chip	Slice usage	RAM usage
SP601	xc6slx16	487 out of 2278 (21%)	RAMB16BWER: 20 out of 32 (62%)
Atlys	xc6slx45	507 out of 6822 (7%)	RAMB16BWER: 20 out of 116 (17%)
S3ESK	xc3s500e	1510 out of 4656 (32%)	RAMB16: 20 out of 20 (100%)

- capture of packets sent and received by the computer, and analysis of the acknowledge latency and “delay” value reported by the emulated FEBs in the data packets (using the wireshark[18] tool)

The tests were performed with the 32 KiB of internal RAM in each FPGA used for packet buffers (32 packets in a single set). All tested boards were connected to the 10/100/1000 Mb/s Ethernet switch D-Link DGS-1008D [19] (see Figure 6.), and each board was assigned a unique MAC address. Results presented in the Table 3 were obtained for parameters of the network congestion avoidance (NCA) algorithm set to $N_{pkt\ update} = 3000$, $T_{high} = 1/16$, $T_{low} = 1/64$, $\alpha_{incr} = 1.25$, $\alpha_{decr} = 0.9375$. The transmission speed was measured during 5 minutes to minimize influence of initial adjustment of the inter-packet delay. For each combination of active boards 12 to 15 measurements were performed, and both average and standard deviation of transmission rate was calculated. Presented results show, that the proposed system reasonably utilizes (over 50%) the bandwidth of the 1 Gb/s Ethernet link available in the receiving system. Probably the achieved throughput could be limited by the speed, at which the application in the receiving system was able to verify the data.

Achieved total throughput is higher for two FEBs with 1 Gb/s interface (Table 3, row 2), than for one such board (Table 3, row 5 and row 6). This probably may be explained by the fact, that the total amount of memory, used to buffer data at the transmitting side is higher in case of two FEBs. The SP601 board provided slightly smaller transmission rate than the Atlys board, even though both boards are equipped with 1Gb/s Ethernet interface, and this result is not explained yet. The SK3E board, equipped with 100 Mb/s Ethernet interface was able to transmit the data with practically maximum speed in all configurations.

The second set of settings ($N_{pkt\ update} = 10000$, $T_{high} = 1/8$, $T_{low} = 1/32$, $\alpha_{incr} = 1.25$, $\alpha_{decr} = 0.75$) of the NCA algorithm was also tested, and appeared to provide reliable transmission of data.

The mean acknowledge latency, measured with the wireshark tool, was equal to 3 μ s, and was significantly lower than the latencies measured in the same computer system for the TCP/IP protocol (see section 2.1).

Table 3: Results of measurement of efficient transmission speed

Active boards	Measured efficient transmission speed [Mb/s]			
	SP601 (1 Gb/s)	Atlys (1 Gb/s)	S3ESK (100 Mb/s)	Total
All boards active	261.42 ($\sigma=14.58$)	284.44 ($\sigma=13.44$)	94.86 ($\sigma=0.36$)	640.72 ($\sigma=26.17$)
SP601 and Atlys active	306.09 ($\sigma=12.03$)	331.44 ($\sigma=18.70$)	–	637.53 ($\sigma=29.97$)
SP601 and SK3E	481.52 ($\sigma=18.92$)	–	94.98 ($\sigma=0.016$)	576.51 ($\sigma=18.93$)
Atlys and SK3E	–	515.27 ($\sigma=19.21$)	94.99 ($\sigma=0.017$)	610.25 ($\sigma=19.21$)
SP601 alone	543.29 ($\sigma=13.40$)	–	–	543.29 ($\sigma=13.40$)
Atlys alone	–	569.95 ($\sigma=14.12$)	–	569.95 ($\sigma=14.12$)
SK3ESK alone	–	–	95.02 ($\sigma=18.99$)	95.02 ($\sigma=18.99$)

Additional tests, with delay introduced in the user space application, proved that the FPGA IP core is able to successfully adapt delay between packets to the processing speed of the receiving system.

The tests have shown that presented system is able to reliably transfer the data from multiple FPGA based FEBs connected via an Ethernet switch to the network interface card in the receiving computer system.

The applied, very simple, network congestion avoidance mechanism appeared to work reliably in case of single, two or three FEBs sending the continuous stream of data, even if the data rates produced by those FEBs differ significantly.

Probably further research may be needed to investigate reliability of our NCA algorithm in more difficult conditions - e.g. in the situation typical for triggered data acquisition systems, where the data rate is fluctuating, reaching the peak value right after the trigger.

5. Availability of the code

The first released version of the code, implementing the described system, has been announced [20] and published [21] in the Usenet newsgroups. The newest version, including files needed to implement it on boards SP601[15], Atlys[16] and Spartan-3E Starter Kit[17] is available on the dedicated website [22].

The licensing information is included in the archive, but generally the whole system is freely available as the open source code, partially under the GPL license, partially under the BSD license, and partially as public domain.

6. Conclusions

The presented system, consisting of the dedicated FPGA IP core, the simple network protocol and the specialized Linux device driver, allows to reliably transmit data from FPGA based Front End Boards (FEBs), to the embedded system via an Ethernet link.

Due to simplicity of the proposed protocol, which leads to simple implementation of the FPGA IP core, and due to minimization of the packet acknowledge latency in the device driver, the system allows to obtain fast, reliable transmission even for small and inexpensive FPGA chips, without necessity to connect them to external RAM.

Data received by the embedded system are placed in the circular buffer, directly available (with memory mapping, assuring minimal data access overhead) for the user space application, which may quickly process and further distribute them. The processing speed may be further increased by use of a multiprocessor embedded system.

In tests the described system allowed to reliably transfer data from 3 FPGA based FEBs, with total throughput up to c.a. 640 Mb/s, via 1Gb/s Ethernet link, using only 32 KiB of internal FPGA memory for data buffering.

The proposed system may be used to concentrate data from FPGA based FEBs to the data processing network, using the standard, inexpensive components, like Ethernet cables and switches, and embedded computer systems equipped with multiple network adapters.

References

- [1] Arria II FPGA Transceiver Overview, <http://www.altera.com/devices/fpga/arria-fpgas/arria-ii-gx/transceivers/aigx-transceivers.html>, 2012. [Online; accessed 15-September-2012].
- [2] Transceivers, <http://www.xilinx.com/products/technology/transceivers/index.htm>, 2012. [Online; accessed 15-September-2012].
- [3] N. Almeida, J. C. Da Silva, R. Alemany, N. Cardoso, J. Varela, Data concentrator card and test system for the CMS ECAL readout. oai:cds.cern.ch:692739, Technical Report CMS-CR-2003-056, 2003.
- [4] W. M. Zabolotny, M. Bluj, K. Bunkowski, M. Gorski, K. Kierzkowski, I. M. Kudla, W. Oklinski, K. T. Pozniak, G. Wrochna, J. Krolikowski, Implementation of the data acquisition system for the Resistive Plate Chamber pattern comparator muon trigger in the CMS experiment, *Measurement Science and Technology* 18 (2007) 2456.
- [5] High Performance TCP/IP on Xilinx FPGA Devices Using the Treck Embedded TCP/IP Stack, http://www.xilinx.com/support/documentation/application_notes/xapp546.pdf, 2004. [Online; accessed 15-September-2012].
- [6] LightWeight IP (lwIP) Application Examples, http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf, 2011. [Online; accessed 15-September-2012].
- [7] IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section One, IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005) (2008) c1 –597.
- [8] IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section Two, IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005) (2008) 1 –790.
- [9] IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section Three, IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005) (2008) 1 –315.
- [10] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, J. Crowcroft, The Use of Forward Error Correction (FEC) in Reliable Multicast, RFC 3453 (Informational), 2002.
- [11] W. M. Zabolotny, I. M. Kudla, K. T. Pozniak, K. Kierzkowski, M. Pietrusinski, G. Wrochna, J. Krolikowski, RPC link box control system for RPC detector in LHC experiment, in: R. S. Romaniuk (Ed.), *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments*, volume 5775 of *Proc. SPIE*, pp. 131–138.

- [12] 10_100_1000 Mbps tri-mode ethernet MAC, http://opencores.org/project,ethernet_tri_mode, 2011. [Online; accessed 15-September-2012].
- [13] C. Benvenuti, Understanding Linux Network Internals, O'Reilly Media, 2006.
- [14] Knoppix version 7.0.3 DVD, http://ftp.uni-kl.de/pub/linux/knoppix-dvd/KNOPPIX_V7.0.3DVD-2012-06-25-EN.iso, 2012. [Online; accessed 15-September-2012].
- [15] Spartan-6 FPGA SP601 Evaluation Kit, <http://www.xilinx.com/products/boards-and-kits/EK-S6-SP601-G.htm>, 2012. [Online; accessed 15-September-2012].
- [16] Atlys™ Spartan-6 FPGA Development Board, <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS>, 2012. [Online; accessed 15-September-2012].
- [17] Spartan-3E Starter Kit, <http://www.xilinx.com/products/boards-and-kits/HW-SPAR3E-SK-US-G.htm>, 2012. [Online; accessed 15-September-2012].
- [18] Wireshark - the world's foremost network protocol analyzer, <http://www.wireshark.org>, 2012. [Online; accessed 15-September-2012].
- [19] 8 Port 10/100/1000Mbps Gigabit Switch DGS-1008D, <http://www.dlink.com/uk/en/business-solutions/switching/unmanaged-switches/desktop/dgs-1008d-8-port-10-100-1000mbps-gigabit-switch>, 2012. [Online; accessed 15-September-2012].
- [20] W. Zabolotny, L3 protocol for transmission from small FPGA to embedded Linux system, Online post to comp.arch.fpga, 2012.
- [21] W. Zabolotny, L3 protocol for data transmission from low resource FPGA to Linux, Online post to alt.sources, also available at <http://ftp.funet.fi/pub/archive/alt.sources/2703.gz>, 2012.
- [22] W. Zabolotny, Layer 3 ethernet protocol for FPGA based systems, http://www.ise.pw.edu.pl/~wzab/fpga_l3_fade, 2012. [Online; accessed 15-September-2012].



Wojciech M. Zabolotny was born in Sucha Beskidzka, Poland in 1966. He received the MSc (1989) and the Ph.D. (1999) in Electronics from the Warsaw University of Technology in Poland, both with honors. Since 1990 he was a research assistant and since 1999 he is an Assistant Professor at the Warsaw University of Technology. His research interests are the distributed data acquisition systems (biomedical and for high energy physics), the embedded systems and programmable logic. He was involved in development of electronic systems for CERN (since 2001), for DESY in Hamburg (2002-2009) and for JET in Culham (since 2010).